Lecture4(part2)

Topics covered:
Arithmetic

Multiplication of signed-operands

- Recall we discussed multiplication of unsigned numbers:
  - Combinatorial array multiplier.
  - Sequential multiplier.
- Need an approach that works uniformly with unsigned and signed (positive and negative 2's complement) n-bit operands.
- Booth's algorithm treats positive and negative 2's complement operands uniformly.

Booth's algorithm

- Booth's algorithm applies uniformly to both unsigned and 2's complement signed integers.
  - Basis of other fast product algorithms.
- Fundamental observation:
  - Division of an integer into the sum of block-1's integers.

Suppose we have a 16-bit binary number: 0110011011110110

This number can be represented as the sum of 4 "block-1" integers:

```
 0110000000000000
 0000011000000000
 0000000011110000
+0000000000000110
 0110011011110110
```

Booth's algorithm (contd..)

Suppose Q is a block-1 integer: Q = 0000000001111000 = 120

Then: X.Q = X.120

Now: 120 = 128 - 8, so that X.Q = X.120 = X.(128-8) = X.128 - X.8

And:

Q = 0000000001111000

128 = 0000000010000000

8 = 0000000000001000

If we label the LSB as 0, then the first 1 in the block of 1's is at position 3 and the last one in the block of 1's is at position 6.
As a result:

$$X.Q = X.120 = X.128 - X.8 = X.2^7 - X.2^3$$

Booth's algorithm (contd..)

Representing Block-1 integers

Q is an *n*-bit block-1 unsigned integer:

- -Bit position 0 is LSB.
- -First 1 is in bit position j
- -Last 1 is in bit position k

Then:

$$Q = 2^{k+1} - 2^{j}$$

$$Q.X = X.(2^{k+1} - 2^{j}) = X.\ 2^{k+1} - X.\ 2^{j}$$

Booth's algorithm (contd..)

Let Q be the block-1 integer: $Q = 0111111111111110$

To form the product $X.Q$ using normal multiplication would involve 14 add/shifts (one for each 1-valued bit in multiplier Q).

Since:

$$Q = 2^{15} - 2^{1}$$
$$X.Q = X.(2^{15} - 2^{1})$$

Product X.Q can be computed as follows:

1. Set the Partial Product (PP) to 0.
2. Subtract $X.2^{1}$ from PP.
3. Add $X.2^{15}$ to PP.

Note that $X.2^{j}$ is equivalent to shifting X left j times.

Booth's algorithm (contd..)

If Q is not a block-1 integer, Q can be decomposed so that it can be represented as a sum of block-1 integers.

Suppose: $Q = 0110011011110110$

Q can be decomposed as:

$$
\begin{array}{rl}
0110000000000000 & = 2^{15} - 2^{13} \\
0000011000000000 & = 2^{11} - 2^{9} \\
0000000011110000 & = 2^{7} - 2^{4} \\
+\underline{0000000000000110} & = 2^{3} - 2^{1} \\
0110011011110110 &
\end{array}
$$

Thus,

$$Q.X = X.(2^{15} - 2^{13} + 2^{11} - 2^{9} + 2^{7} - 2^{4} + 2^{3} - 2^{1})$$

Booth's algorithm (contd..)

Inputs: n-bit multiplier Q
n-bit multiplicand x
2n-bit current Partial Product (PP) initially set to 0.
(Upper half of PP is bits n-1 through n)
Q has an added '0' bit attached to the LSB (Q has n+1 bits).

Algorithm: For every bit in Q:

1. Examine the bit and its neighbor to the immediate right.
   If the bit pair is:
   - 00 – do nothing.
   - 01 – Add the multiplicand to the upper half of PP.
   - 10 – Sub the multiplicand from the upper half of PP.
   - 11 – Do nothing.
2. Shift the PP right by one bit, extending the sign bit.

Signed multiplication

						1	0	0	1	1	(−13)
					×	0	1	0	1	1	(+11)
	1	1	1	1	1	1	0	0	1	1	
	1	1	1	1	1	0	0	1	1		
Sign extension is shown in blue	0	0	0	0	0	0	0	0			
	1	1	1	0	0	1	1				
	0	0	0	0	0	0					
	1	1	0	1	1	1	0	0	0	1	(−143)

Figure 6.8 Sign extension of negative multiplicand.

Booth Multiplier Encoding

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Figure 6.12 Booth multiplier recoding table.

0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0

⇓

0 +1 −1 +1 0 −1 0 +1 0 0 −1 +1 −1 +1 0 −1 0 0

Figure 6.10 Booth recoding of a multiplier.

Booth Multiplication with negative multiplier

```
   0 1 1 0 1   (+13)           0  1  1  0  1
 x 1 1 0 1 0   (-6)    ⇒       0 -1 +1 -1  0
 -----------                  ---------------
                     0 0 0 0 0 0 0 0 0 0
                     1 1 1 1 1 0 0 1 1
                     0 0 0 0 1 1 0 1
                     1 1 1 0 0 1 1
                     0 0 0 0 0 0
                     -------------------
                     1 1 1 0 1 1 0 0 1 0   (-78)
```

Fast Multiplication

Bit-pair encoding of multiplier

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	$0 \times M$
0	0	1	$+1 \times M$
0	1	0	$+1 \times M$
0	1	1	$+2 \times M$
1	0	0	$-2 \times M$
1	0	1	$-1 \times M$
1	1	0	$-1 \times M$
1	1	1	$0 \times M$

(b) Table of multiplicand selection decisions

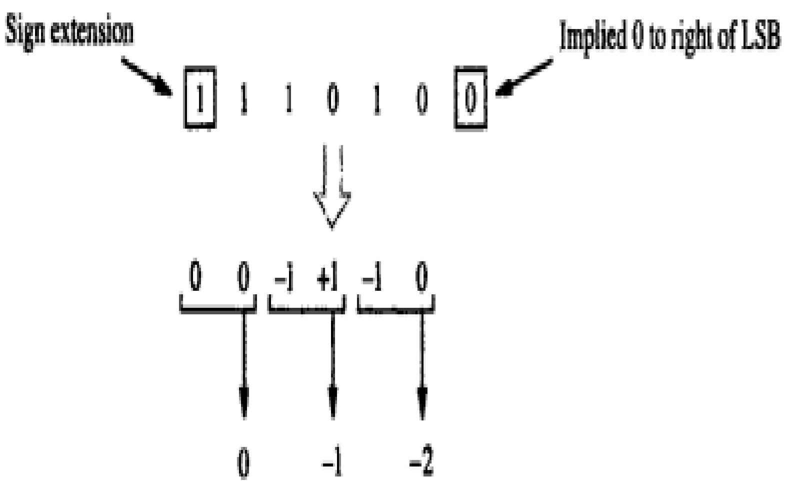


Figure 6.14 Multiplier bit-pair recoding.

Booth encoding versus Bit-pair encoding of multiplier

```
            0  1  1  0  1   (+13)
         ×  1  1  0  1  0   (−6)
         ----------------

                  ⇓

            0  1  1  0  1
            0 −1 +1 −1  0
         ----------------
0  0  0  0  0  0  0  0  0  0
1  1  1  1  1  0  0  1  1
0  0  0  0  1  1  0  1
1  1  1  0  0  1  1
0  0  0  0  0  0
----------------------------
1  1  1  0  1  1  0  0  1  0   (−78)
```

Using Booth encoding

```
          0  1  1  0  1
          0    −1    −2
         ---------------
1 1 1 1 1 0  0  1  1  0
1 1 1 1 0 0  1  1
0 0 0 0 0 0
-----------------------
1 1 1 0 1 1  0  0  1  0
```

Using bit-pair encoding

Unsigned division

- Division is a more tedious process than multiplication.
- For the unsigned case, there are two standard approaches:
  1.) Restoring division. 2.) Non restoring division.

```
                21   quotient
divisor  13 ) 274    dividend
              26
              --
               14
               13
               --
                1
```

Try dividing 13 into 2.
Try dividing 13 into 26.

```
            10101
1101 ) 10001001
       01101
       -----
         10000
          1101
         -----
           1110
           1101
           ----
              1
```

Try dividing 1101 into 1, 10, 100, 1000 and 10001.

Restoring division

How do we know when the divisor has gone into part of the dividend correctly?

```
           10101
      ___________
1101 ) 10001001
       01101
       -----
         10000
          1101
         -----
           1110
           1101
          -----
              1
```

Subtract 1101 from 1, result is negative
Subtract 1101 from 10, result is negative.
Subtract 1101 from 100, result is negative
Subtract 1101 from 1000, result is negative.
Subtract 1101 from 10001, result is positive.

Restoring division

Strategy for unsigned division:

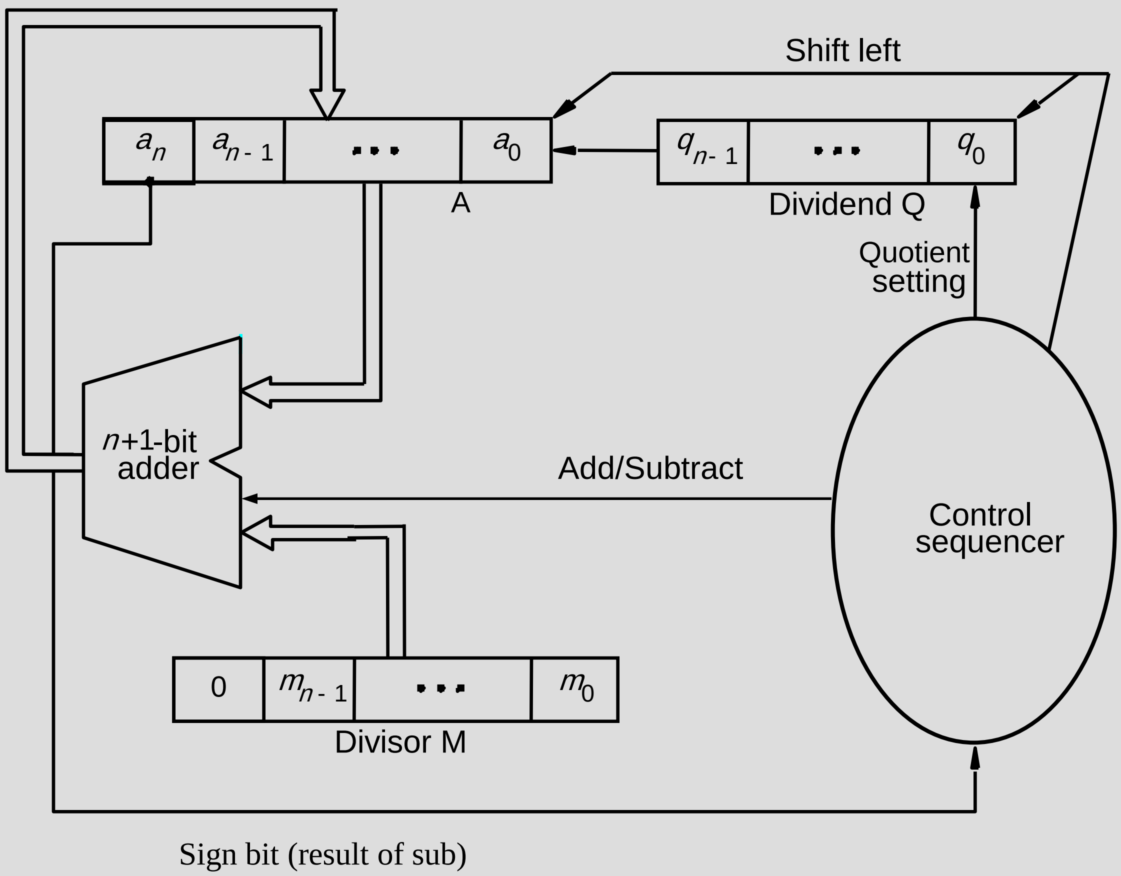
Shift the dividend one bit at a time starting from MSB into a register.
Subtract the divisor from this register.
If the result is negative (“didn’t go”):

- Add the divisor back into the register.
- Record 0 into the result register.

If the result is positive:

- Do not restore the intermediate result.
- Set a 1 into the result register.

Restoring division (contd..)



Set Register A to 0.
Load dividend in Q.
Load divisor into M.
Repeat n times:
- *Shift A and Q left one bit.*
- *Subtract M from A.*
- *Place the result in A.*
- *If sign of A is 1, set q_0 to 0 and add M back to A.*
  Else set q_0 to 1.

End of the process:
- *Quotient will be in Q.*
- *Remainder will be in A.*

Restoring division (contd..)

$$\begin{array}{r} 10 \\ 11 \overline{)\,1000} \\ \underline{11} \\ 10 \end{array}$$

	A	Dividend	
Initially	0 0 0 0 0	1 0 0 0	
M	0 0 0 1 1		
Shift	0 0 0 0 1	0 0 0	
Subtract	1 1 1 0 1		First cycle
Set q_0	1 1 1 1 0		
Restore	1 1		
	0 0 0 0 1	0 0 0 0	
Shift	0 0 0 1 0	0 0 0	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1		Second cycle
Restore	1 1		
	0 0 0 1 0	0 0 0 0	
Shift	0 0 1 0 0	0 0 0	
Subtract	1 1 1 0 1		
Set q_0	0 0 0 0 1		Third cycle
Shift	0 0 0 1 0	0 0 0 1	
Subtract	1 1 1 0 1	0 0 1	
Set q_0	1 1 1 1 1		Fourth cycle
Restore	1 1		
	0 0 0 1 0	0 0 1 0	
	Remainder	Quotient	

Non-restoring division

Restoring division can be improved using non-restoring algorithm

The effect of restoring algorithm actually is:
If A is positive, we shift it left and subtract M, that is compute $2A-M$
If A is negative, we restore it ($A+M$), shift it left, and subtract M, that is, $2(A+M) - M = 2A+M$.
Set q_0 to 1 or 0 appropriately.

The restoring-division algorithm can be improved by avoiding the need for restoring A after an unsuccessful subtraction. Subtraction is said to be unsuccessful if the result is negative.

Non-restoring algorithm is:
Set A to 0.
Repeat n times:
1- If the sign of A is positive:
Shift A and Q left and subtract M. Set q0 to 1.
Else if the sign of A is negative:
Shift A and Q left and add M. Set q0 to 0.
End of loop
2-If the sign of A is 1, add A to M.

Non-restoring division (contd..)

```
     10
11 ) 1000
      11
      10
```

Initially	0 0 0 0 0	1 0 0 0	First cycle
	0 0 0 1 1		
Shift	0 0 0 0 1	0 0 0	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0	0 0 0 0	
Shift	1 1 1 0 0	0 0 0	Second cycle
Add	0 0 0 1 1		
Set q_0	1 1 1 1 1	0 0 0 0	
Shift	1 1 1 1 0	0 0 0	Third cycle
Add	0 0 0 1 1		
Set q_0	0 0 0 0 1	0 0 0 1	
Shift	0 0 0 1 0	0 0 1	Fourth cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1	0 0 1 0	
		Quotient	

Add	1 1 1 1 1	Restore remainder
	0 0 0 1 1	
	0 0 0 1 0	
	Remainder	